Conditional and Repetitive Processing

This chapter covers the following topics:

- Conditional Processing
- Repetitive Processing
- Defining the Relation
- Defining the Consequence
- Logical Operators

When you use the .EP instruction in an .IF construction, formatting is always stopped.

Conditional Processing

.IF - Start Conditional Processing

```
.IF relation
```

Conditional processing is started using the .IF instruction. The .IF instruction defines a relation (see *Defining the Relation*).

When the result of the relation is true, the defined consequence is processed (see *Defining the Consequence*). When the result of the relation is false, the defined consequence is skipped.

When you use the .IF instruction, you can either define a simple consequence (see below) or a compound consequence (see the .TH instruction).

The Simple Consequence

A simple consequence consists of either a single line of text or an instruction after the .IF instruction. In the following example, the output "Congratulations" is the simple consequence:

```
.IF &age = 50
Congratulations
```

The simple consequence terminates the conditional statement.

Note:
You should not use the .EI instruction (End-If) after a simple consequence.

You can also use the logical operators .AN or .OR (see *Logical Operators*). In the following example, the output "Dear Mr." is the simple consequence:

```
.IF &age = 50;.AN &sex = M
Dear Mr.
```

If the consequence is an instruction, you can define it in the same line as the .IF instruction. In the following example, the instruction .NP is the simple consequence:

```
.IF &$LC < 5;.NP
```

.TH - Define a Compound Consequence (Then)

```
.TH
```

A compound consequence consists of several text lines and/or instructions after the .IF instruction. It must be introduced with the .TH instruction and terminated with the .EI instruction. For example:

```
.IF &$DN = Monday
.TH
.NP
&$MN &$DA., &$YE
.IL 1
This is the start of a new week.
.EI
```

In the above example, the instructions and text between .TH and .EI form the compound consequence (start a new page, print the current date, insert one blank line and output the specified text).

The instructions .TH and .EI form a pair, i.e. when you use the .TH instruction, it must be followed by a matching .EI instruction.

Note:
It is possible to nest relations. The maximum nesting depth is 10.

.EL - Define an Alternative (Else)

```
.EL
```

When you use the .TH instruction, you can also define an alternative. If the result of the defined .IF relation is false, the alternative is processed (instead of the consequence after the .TH instruction).

An alternative can consist of several text lines and/or instructions after the .EL instruction. For example:

```
.IF &sex = M
.TH
Mr
.EL
Ms
.EI
```

If the variable &sex in the above example does not have the value M, the alternative between the instructions .EL and .EI (include "Ms" in the formatted text) is processed.

The rules which apply to the consequence (see *Defining the Consequence*), also apply to the alternative.

Note:
When you use the .EL instruction, you must also use the instructions .TH and .EI.

.EI - End Conditional Processing (End-If)

```
.EI
```

The .EI instruction is used to indicate the end of a compound consequence (including the alternative) which has been introduced using the .TH instruction.

Note:
You must use the .EI instruction when you have previously used the .TH instruction.

The whole .IF construction (i.e. all specifications between .IF and .EI) must be specified within the same document.

Repetitive Processing

.WH - Start While-Loop

```
.WH relation
```

Repetitive processing is started using the .WH instruction. The .WH instruction defines a relation (see *Defining the Relation*).

The relation is followed by a consequence which can consist of several text lines and/or instructions that are to be repeated (see *Defining the Consequence*).

The end of the loop must be indicated by the .EW instruction.

As long as the result of the relation is true, the defined consequence is processed. If the result of the relation is false, the While-loop is terminated and processing continues after the .EW instruction.

The following is an example for a simple While-loop:

```
.SV counter=0
.WH &counter <= 3;.** Loop 3 times.
.SV counter=&counter+1;.** Increment the value of the variable &counter.
Number &counter
.EW
&counter was the last number.
```

The above instructions cause the following formatted output:

```
Number 1
Number 2
Number 3
3 was the last number.
```

The result of the relation must change from true to false. If the result is always true, Con-form loops endlessly. If the result of the relation is false the first time the .WH instruction occurs, it is skipped without being processed at all.

Note:
It is not possible to nest While-loops.

Processing a List of Variables with Similar Names

When you use the .WH instruction, you can process a list of variables which have similar names that end with a number.

To do so, you must define an additional numeric variable and increment its value within the loop. Then you must combine the beginning of the name variable with the number variable as shown in the example below:

```
.SV name1=James
.SV name2=Lars
.SV name3=Jason
.SV name4=Kirk
.SV number=1;.** Define the additional numeric variable.
.WH &number <= 4
.** Combine the beginning of the variable &name with the variable &number:
&name&number
.SV number=&number+1;.** Increment the value of the variable &number.
.EW
```

The above instructions cause the following formatted output:

```
James
Lars
Jason
Kirk
```

.WX - Exit from While-Loop

```
.WX
```

The .WX instruction can be used to exit from a While-loop before its end is reached and to continue processing after the .EW instruction. In the following example, the occurrence of the name "Lars" terminates the execution of the While-loop:

```
.SV name1=James
.SV name2=Lars
.SV name3=Jason
.SV name4=Kirk
.SV number=1
.WH &number <= 4
&name&number
.IF &name&number = Lars
.WX
.SV number=&number+1;
.EW
.SL 1
&name&number was the last name in the loop.
```

The above instructions cause the following formatted output:

```
James
Lars

Lars was the last name in the loop.
```

.EW - End While-Loop

```
.EW
```

The .EW instruction must be used in conjunction with the .WH instruction. It indicates the end of the While-loop.

It is not possible to conditionally suppress the processing of the .EW instruction by using it in an .IF construction.

Defining the Relation

Both the .IF and .WH instructions must be followed by a relation on the same line. A relation consists of two values separated by a relational operator.

If you want to define more than one relation with the .IF or .WH instruction, you must separate the relations by the logical operators .AN or .OR (see *Logical Operators*).

Relational Operators

You must specify one of the following relational operators between the two values of a .IF or .WH instruction:

Operator		**Description**
=	EQ	Equal to
<>	NE	Not equal to
<	LT	Less than
<=	LE	Less than or equal to
>	GT	Greater than
>=	GE	Greater than or equal to
?		Substring
*		Subset

Most relational operators have alternative forms as shown in the table above. For example, the relational operator "Equal to" can be defined by using either the equal sign (=) or the abbreviation EQ. Thus, the following two instructions are identical:

```
.IF &age = 50
.IF &age EQ 50
```

You must include at least one blank space before and after the relational operator.

The first value (to the left of the relational operator) must be a variable (see *Variables*).

The second value (to the right of the relational operator) can either be a variable or a constant (for example, a number or name).

In some cases, it is not necessary to specify the second value. For example, if you want to process a consequence only when the value of a variable is not blank, you can specify:

```
.IF &street NE;.TH;.BR
&street
.EI
```

Enclosing a Constant Within Apostrophes

If a constant includes blank spaces or commas, the constant must be enclosed within apostrophes. For example:

```
.IF &company <> 'Software AG'
```

It is possible to include an apostrophe in a constant which is enclosed within apostrophes by repeating the apostrophe. For example:

```
.IF &company <> 'Software AG''s'
```

It is possible to include apostrophes within a parameter which is not enclosed within apostrophes. For example:

```
.IF &company <> AG's
```

Any number of apostrophes may appear within the constant as long as the first character is not an apostrophe. For example:

```
.IF &company <> Software'AG's
```

Substring (?)

The substring operator ? is used to determine whether the left string is a substring of the right string.

In the following example, the left string is a subset of the right string. Therefore, the relation is considered as true.

```
.IF rat ? scratch
```

In the following example, the left string is *not* a subset of the right string. Therefore, the relation considered as false.

```
.IF cat ? scratch
```

Note:
When you use the substring operator, the first value need not necessarily be a variable. It can also be a constant.

Subset (*)

The subset operator * is used to determine whether each character which occurs in the left string also occurs in the right string.

In the following examples, all characters of the left string also occur in the right string. Therefore, all relations are considered as true.

```
.IF a * scratch
.IF chat * scratch
.IF ttt * scratch
```

In the following example, *not* all characters of the left string occur in the right string. Therefore, the relation is considered as false.

```
.IF mat * scratch
```

Note:
When you use the subset operator, the first value need not necessarily be a variable. It can also be a constant.

Defining the Consequence

A consequence can be an instruction and/or text. It is processed when the result of a relation is true.

If the consequence is an instruction, you can specify it in the same line as the .IF or .WH instruction, or in the same line as the last .AN or .OR operator - separated by the instruction separator character (initially, this is the semicolon). For example:

```
.IF &age > 60;.TH
```

If the consequence is text, it must start in a separate line. For example:

```
.IF &age = 50
Congratulations
```

Note:
The above rules also apply for an alternative that is defined using the .EL instruction.

Logical Operators

The logical operators .AN and .OR can be used with the .IF and .WH instructions.

These operators are treated as separate instructions. Therefore, they must be separated from the preceding relation by the instruction separator character (initially, this is the semicolon). For example:

```
.IF &age = 50;.AN &sex = M
```

When you specify several logical operators in the same line, they are processed from left to right.

However, you can also specify the .AN or .OR instruction in a separate line. For example:

```
.IF &age = 50
.AN &sex = M
```

.AN - And

.AN *relation*

When you use the logical operator .AN, *all* relations must be true. Otherwise the defined consequence is not processed. For example:

```
.SV age=65
.SV sex=M
.IF &age GE 65;.AN &sex = F
Dear Grandma
```

Since only the first relation is true in the above example, the words "Dear Grandma" are not output.

.OR - Or

.OR *relation*

When you use the logical operator .OR, at least one of the relations must be true. Otherwise the defined consequence is not processed. For example:

```
.SV age=65
.SV sex=M
.IF &age GE 65;.OR &sex = F
Dear Grandma
```

In the above example, only the first relation is true. However, the words "Dear Grandma" are output, since it is only required that one of the two relations is true.